ELG5255 Applied Machine Learning
Group Assignment 1

Group 9

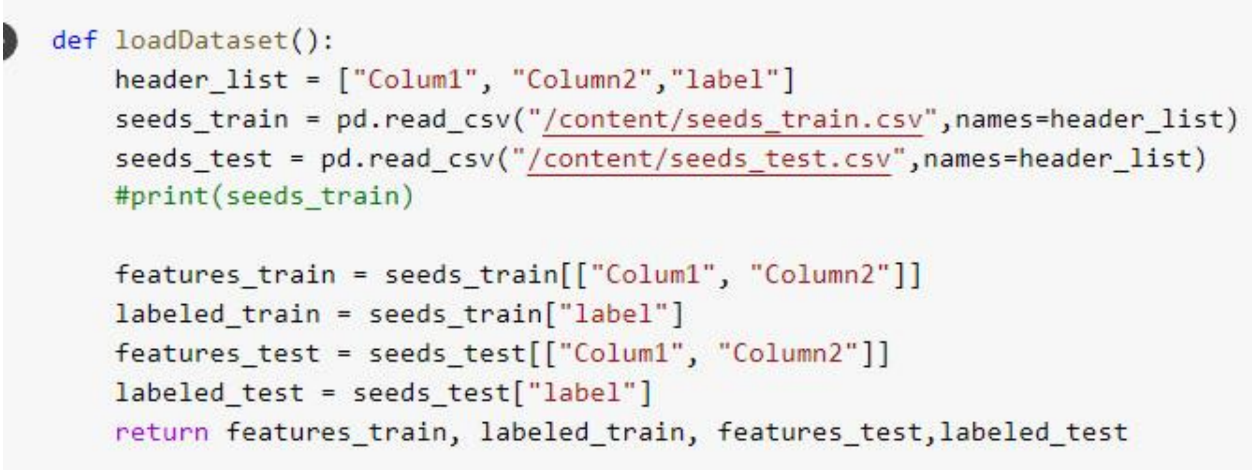
1. Wael Mofreh Ismail Ali Elsharkawy – 300273116
2. Sameh Mohamed Ismail Deabes - 300273126

Goal :

Our goal is to build model give me high performance and accuracy and make enhance to the model by using more than 1 algorism and make aggregation to the prediction and to improve the model

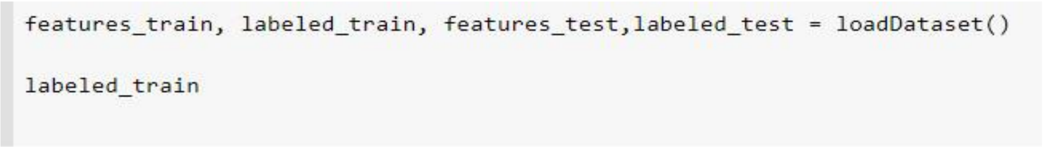
Dataset :

1. The data is coming cleaned and was made a process on it   and the data suitable to word with it directly with model.
2. Dataset is already divided into the training and testing and that helps to could make detection the data belong to class (1,2,3) ((Kama, Rosa and Canadian))

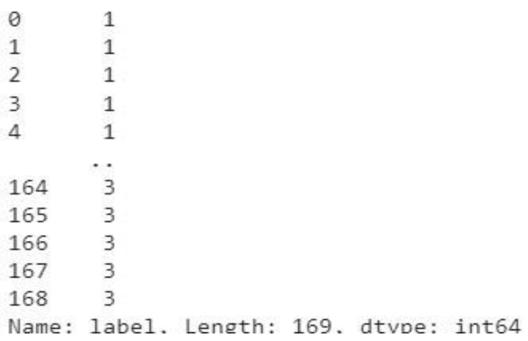


```
def loadDataset():
    header_list = ["Colum1", "Column2","label"]
    seeds_train = pd.read_csv("/content/seeds_train.csv",names=header_list)
    seeds_test = pd.read_csv("/content/seeds_test.csv",names=header_list)
    #print(seeds_train)

    features_train = seeds_train[["Colum1", "Column2"]]
    labeled_train = seeds_train["label"]
    features_test = seeds_test[["Colum1", "Column2"]]
    labeled_test = seeds_test["label"]
    return features_train, labeled_train, features_test,labeled_test
```

2. Represent data to see the feature and lapels



```
features_train, labeled_train, features_test,labeled_test = loadDataset()

labeled_train
```



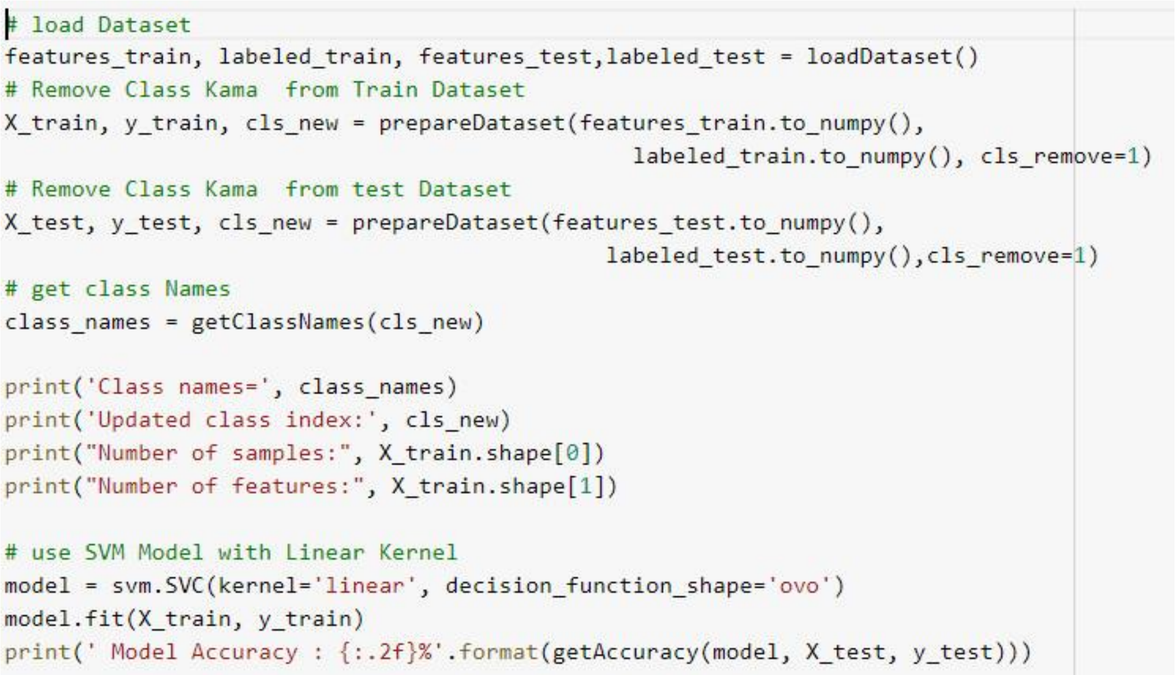
```
0      1
1      1
2      1
3      1
4      1
      ..
164    3
165    3
166    3
167    3
168    3
Name: label, Length: 169, dtype: int64
```

3. We make some function that will use it to make visualization to our model
4. Firest model work on data binary after remove first class (Kama), compare performance of Perceptron and SVM on testing set and get the confusion matrix and

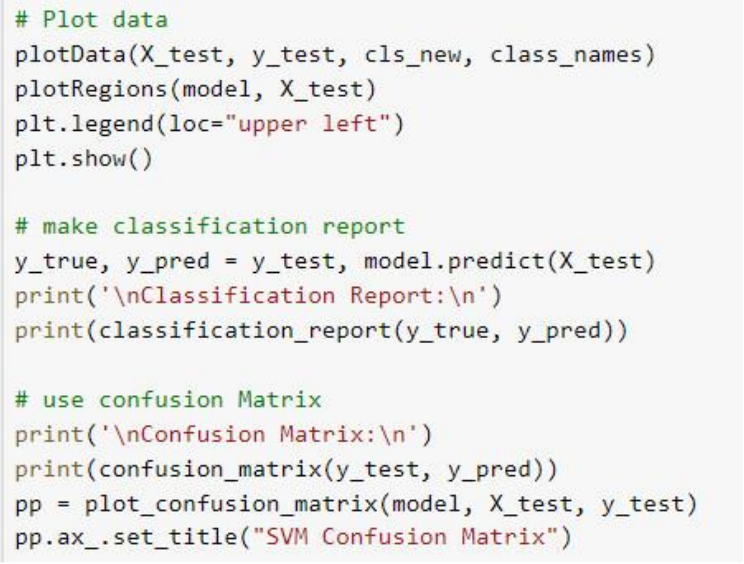
SVM Model

Output is :The accuracy of the model is 100% that maens the model is predicted all values correctly.

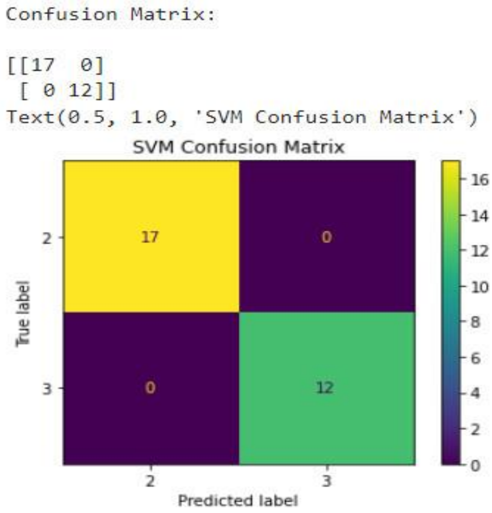
Code and output:



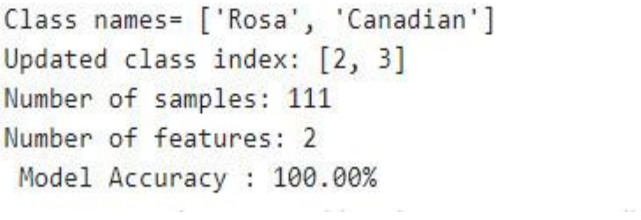
```
# load Dataset
features_train, labeled_train, features_test,labeled_test = loadDataset()
# Remove Class Kama  from Train Dataset
X_train, y_train, cls_new = prepareDataset(features_train.to_numpy(),
                                             labeled_train.to_numpy(), cls_remove=1)
# Remove Class Kama  from test Dataset
X_test, y_test, cls_new = prepareDataset(features_test.to_numpy(),
                                           labeled_test.to_numpy(),cls_remove=1)
# get class Names
class_names = getClassNames(cls_new)

print('Class names=', class_names)
print('Updated class index:', cls_new)
print("Number of samples:", X_train.shape[0])
print("Number of features:", X_train.shape[1])

# use SVM Model with Linear Kernel
model = svm.SVC(kernel='linear', decision_function_shape='ovo')
model.fit(X_train, y_train)
print(' Model Accuracy : {:.2f}%'.format(getAccuracy(model, X_test, y_test)))
```



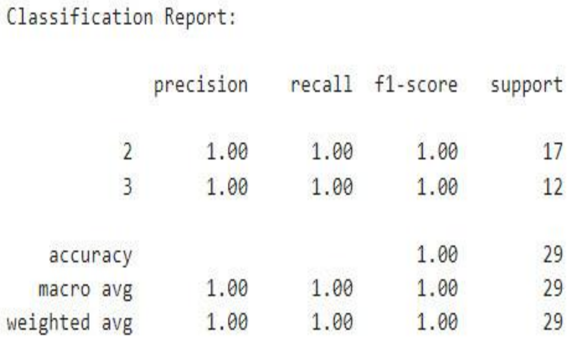
```
# Plot data
plotData(X_test, y_test, cls_new, class_names)
plotRegions(model, X_test)
plt.legend(loc="upper left")
plt.show()

# make classification report
y_true, y_pred = y_test, model.predict(X_test)
print('\nClassification Report:\n')
print(classification_report(y_true, y_pred))

# use confusion Matrix
print('\nConfusion Matrix:\n')
print(confusion_matrix(y_test, y_pred))
pp = plot_confusion_matrix(model, X_test, y_test)
pp.ax_.set_title("SVM Confusion Matrix")
```



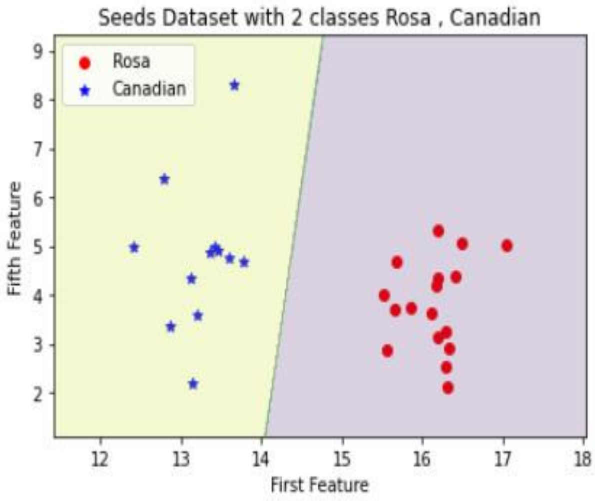
```
Confusion Matrix:

[[17  0]
 [ 0 12]]
Text(0.5, 1.0, 'SVM Confusion Matrix')
```



```
Class names= ['Rosa', 'Canadian']
Updated class index: [2, 3]
Number of samples: 111
Number of features: 2
 Model Accuracy : 100.00%
```

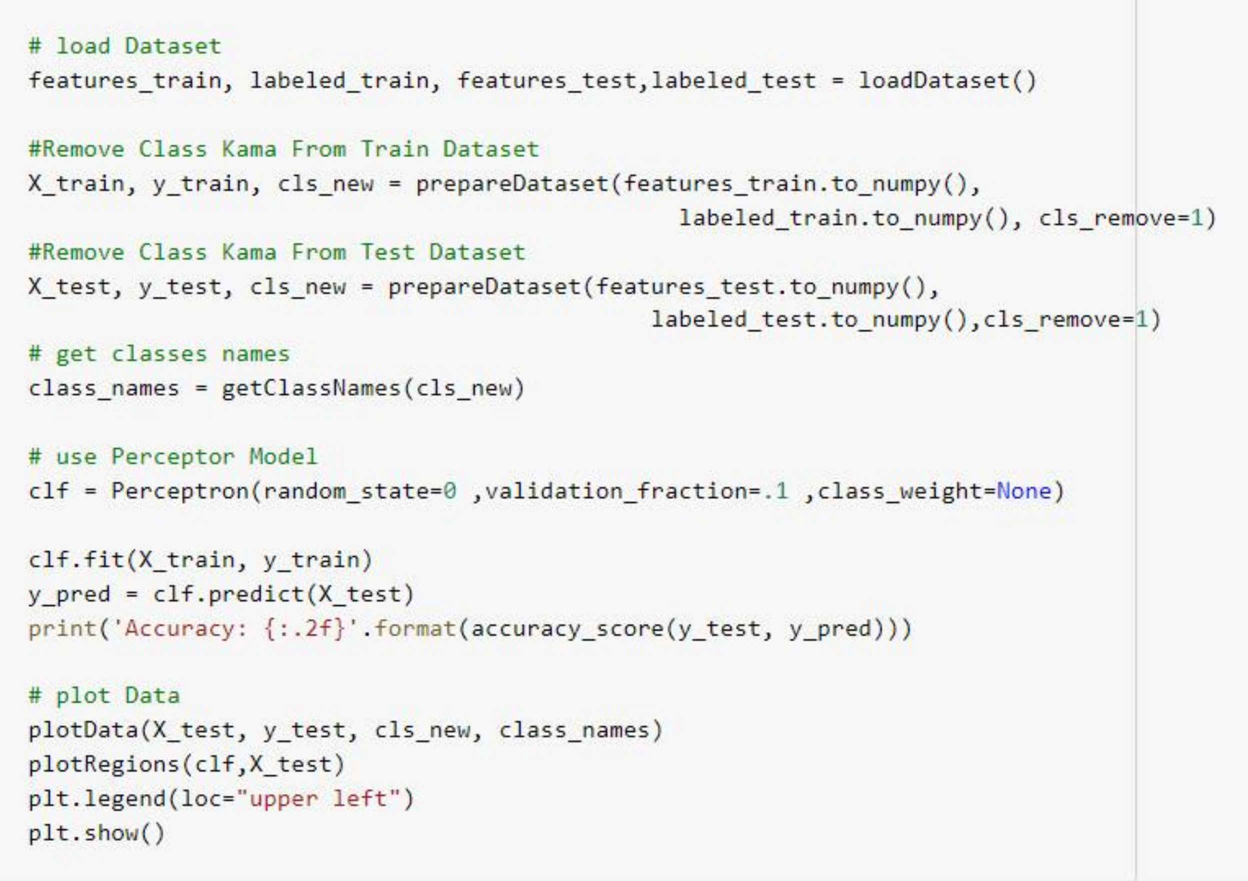


```
Classification Report:

              precision    recall  f1-score   support

           2       1.00      1.00      1.00        17
           3       1.00      1.00      1.00        12

    accuracy                           1.00        29
   macro avg       1.00      1.00      1.00        29
weighted avg       1.00      1.00      1.00        29
```

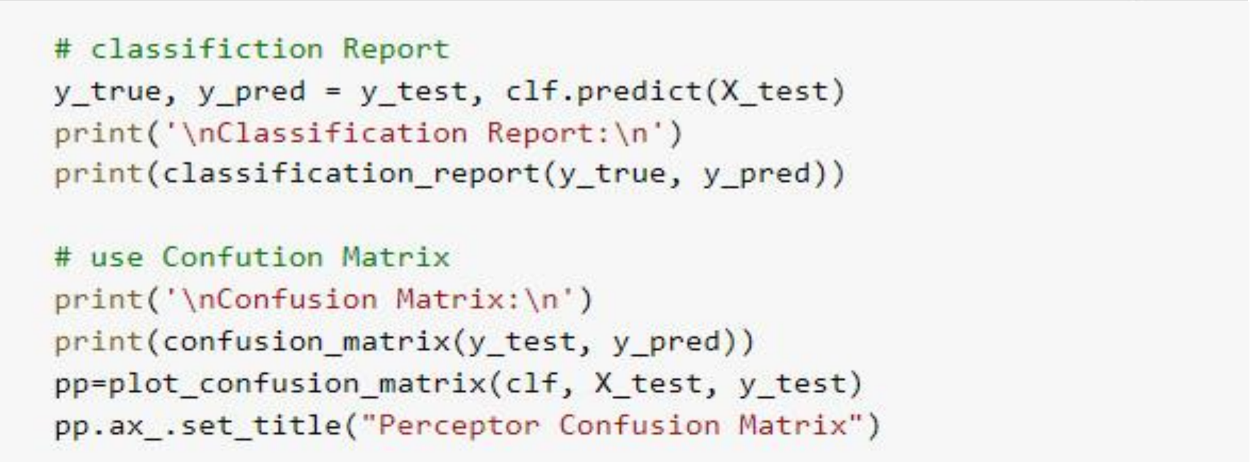


Perceptron Model:

we work on the perceptron to get the accuracy of the model to make classification in class 2,3 (**Rosa and Canadian)** the model give us accuracy 83%



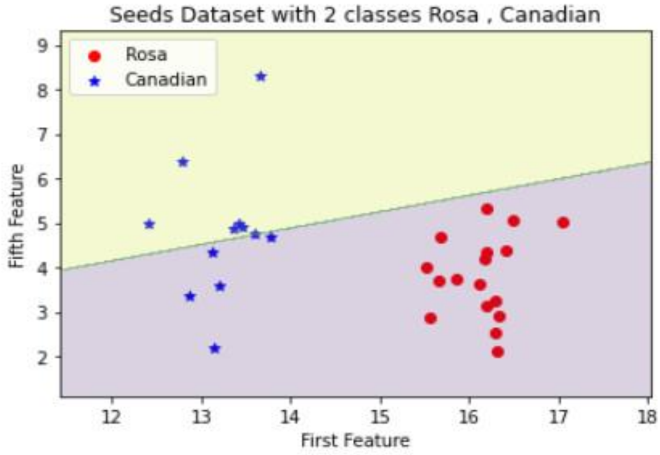
```
# load Dataset
features_train, labeled_train, features_test,labeled_test = loadDataset()

#Remove Class Kama From Train Dataset
X_train, y_train, cls_new = prepareDataset(features_train.to_numpy(),
                                               labeled_train.to_numpy(), cls_remove=1)
#Remove Class Kama From Test Dataset
X_test, y_test, cls_new = prepareDataset(features_test.to_numpy(),
                                             labeled_test.to_numpy(),cls_remove=1)
# get classes names
class_names = getClassNames(cls_new)

# use Perceptor Model
clf = Perceptron(random_state=0 ,validation_fraction=.1 ,class_weight=None)

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print('Accuracy: {:.2f}'.format(accuracy_score(y_test, y_pred)))

# plot Data
plotData(X_test, y_test, cls_new, class_names)
plotRegions(clf,X_test)
plt.legend(loc="upper left")
plt.show()
```

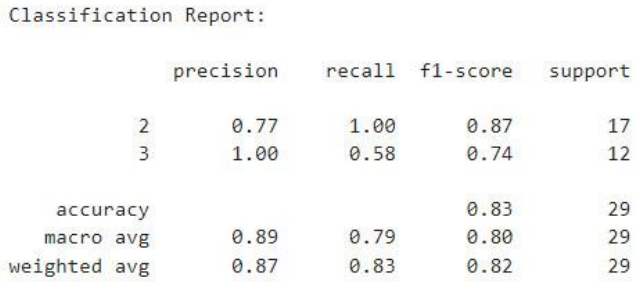


```
# classifiction Report
y_true, y_pred = y_test, clf.predict(X_test)
print('\nClassification Report:\n')
print(classification_report(y_true, y_pred))

# use Confution Matrix
print('\nConfusion Matrix:\n')
print(confusion_matrix(y_test, y_pred))
pp=plot_confusion_matrix(clf, X_test, y_test)
pp.ax_.set_title("Perceptor Confusion Matrix")
```

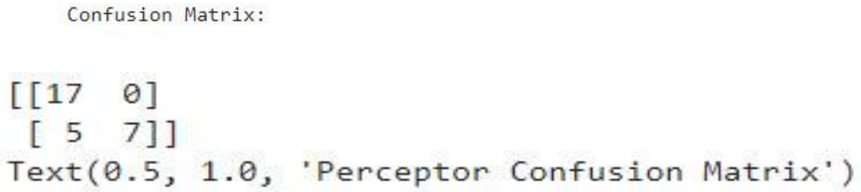
Output of the model :

```
Accuracy: 0.83
```

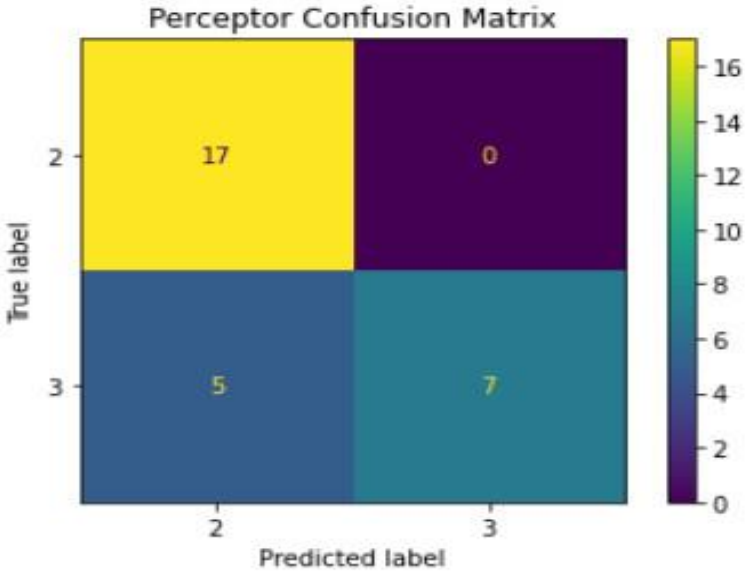




```
Classification Report:

              precision    recall  f1-score   support

           2       0.77      1.00      0.87        17
           3       1.00      0.58      0.74        12

    accuracy                           0.83        29
   macro avg       0.89      0.79      0.80        29
weighted avg       0.87      0.83      0.82        29
```



```
Confusion Matrix:

[[17  0]
 [ 5  7]]
Text(0.5, 1.0, 'Perceptor Confusion Matrix')
```



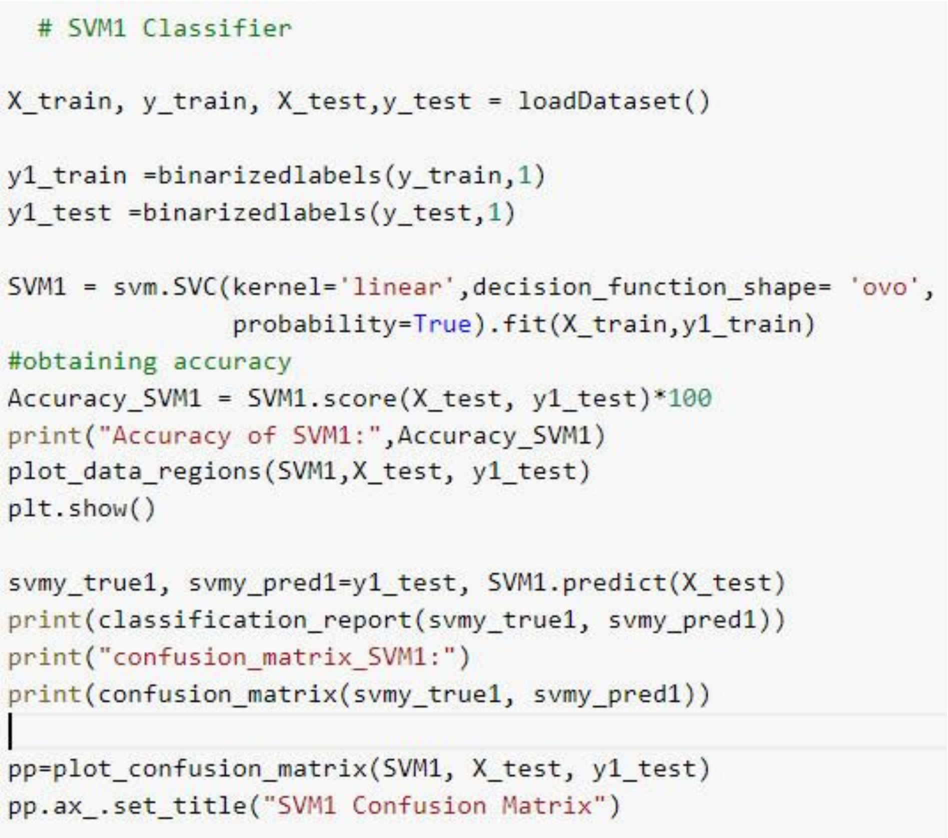
OVR SVM:

Made binarize labels for Train labels and Test labels by using `binarizedlabels` function that Obtain the binarized labels.1 for positive class, -1 for negative class (OvR) as Follows :

```
#Obtain the binarized label (1 for positive class, -
1 for negative class)
def binarizedlabels(y_labels,myclass):
    y_labels = np.copy(y_labels)
    for index, label in enumerate(y_labels):
        if y_labels[index] == myclass:
            y_labels[index]=1
        else:
            y_labels[index]=-1
    return y_labels
```

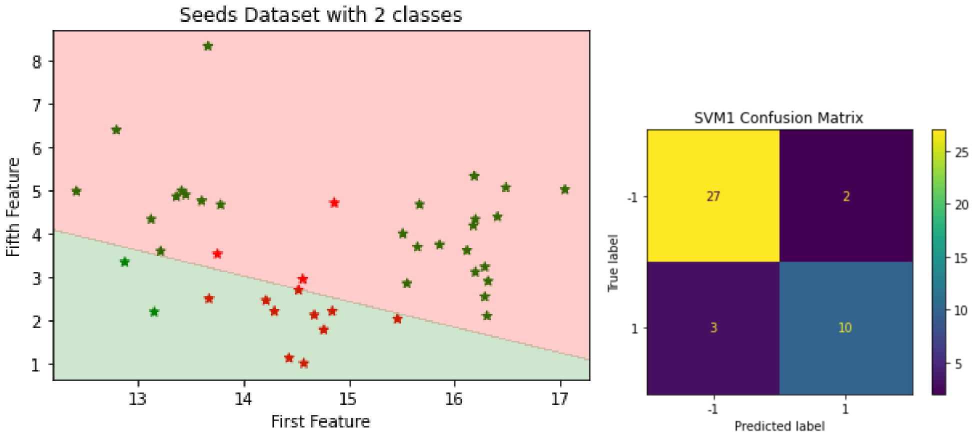
SVM OVR

SVM First Classifier Plot decision boundary and Confusion Matrix and Accuracy: 88.095% we get the accuracy of the model and compare with other data

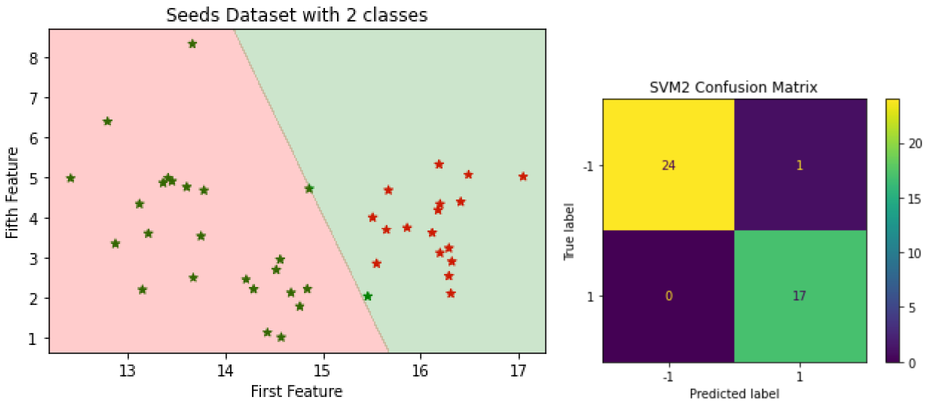


```
  # SVM1 Classifier

X_train, y_train, X_test,y_test = loadDataset()

y1_train =binarizedlabels(y_train,1)
y1_test =binarizedlabels(y_test,1)

SVM1 = svm.SVC(kernel='linear',decision_function_shape= 'ovo',
               probability=True).fit(X_train,y1_train)
#obtaining accuracy
Accuracy_SVM1 = SVM1.score(X_test, y1_test)*100
print("Accuracy of SVM1:",Accuracy_SVM1)
plot_data_regions(SVM1,X_test, y1_test)
plt.show()

svmy_true1, svmy_pred1=y1_test, SVM1.predict(X_test)
print(classification_report(svmy_true1, svmy_pred1))
print("confusion_matrix_SVM1:")
print(confusion_matrix(svmy_true1, svmy_pred1))

pp=plot_confusion_matrix(SVM1, X_test, y1_test)
pp.ax_.set_title("SVM1 Confusion Matrix")
```

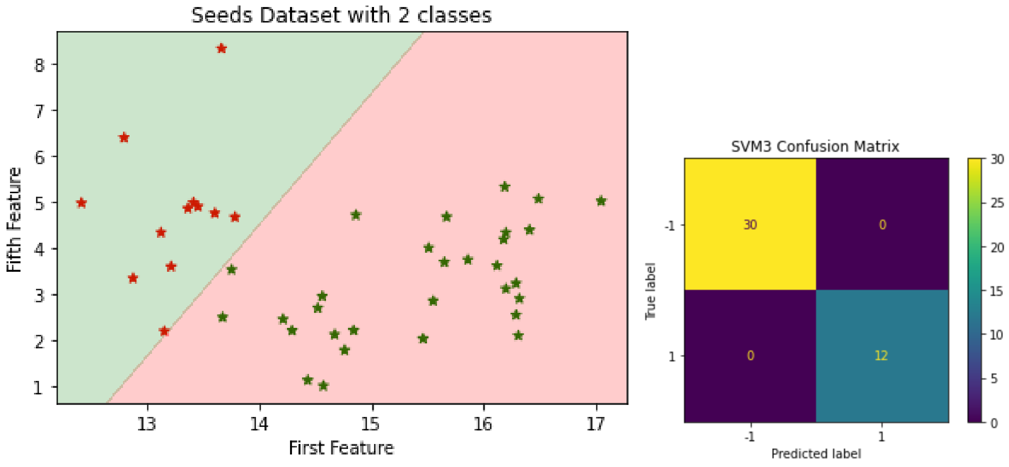
SVM First Classifier Plot decision boundary and Confusion Matrix and Accuracy: 88.095%



SVM Second Classifier Plot decision boundary and Confusion Matrix and Accuracy: 97.61%

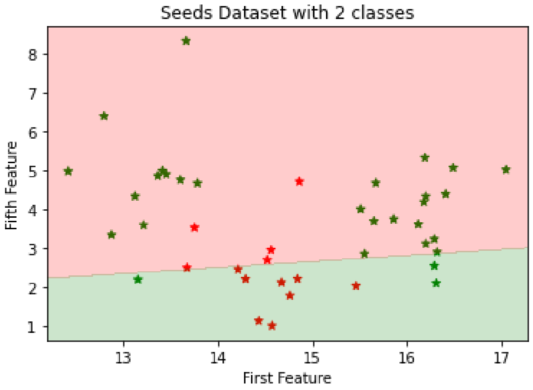


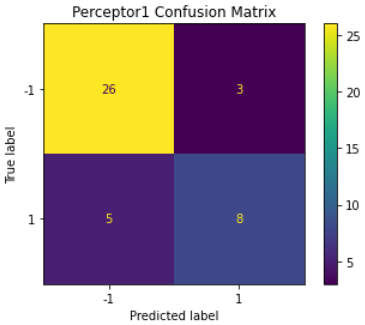
SVM Third Classifier Plot decision boundary and Confusion Matrix and Accuracy: 100%



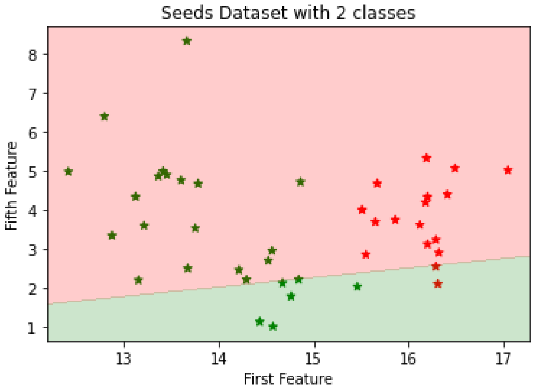
OVR Perceptron

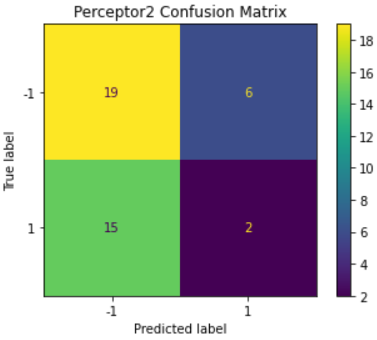
Perceptron First Classifier Plot decision boundary and Confusion Matrix and Accuracy: 81%



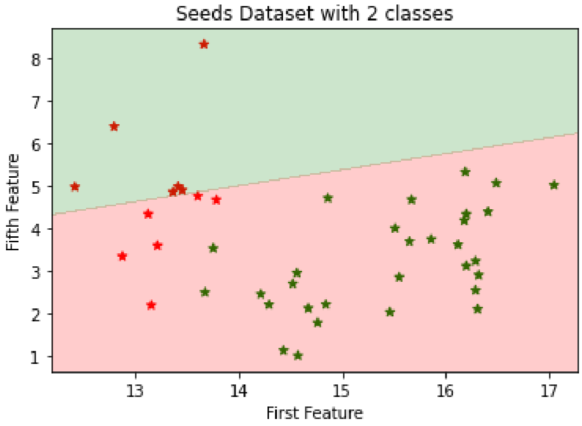


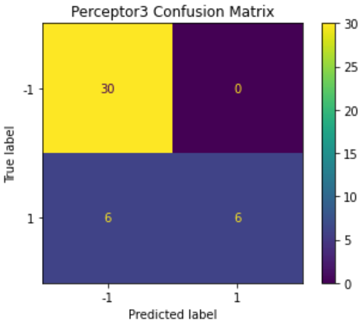
Perceptron Second Classifier Plot decision boundary and Confusion Matrix and Accuracy: 50%





Perceptron Third Classifier Plot decision boundary and Confusion Matrix and Accuracy: 86%

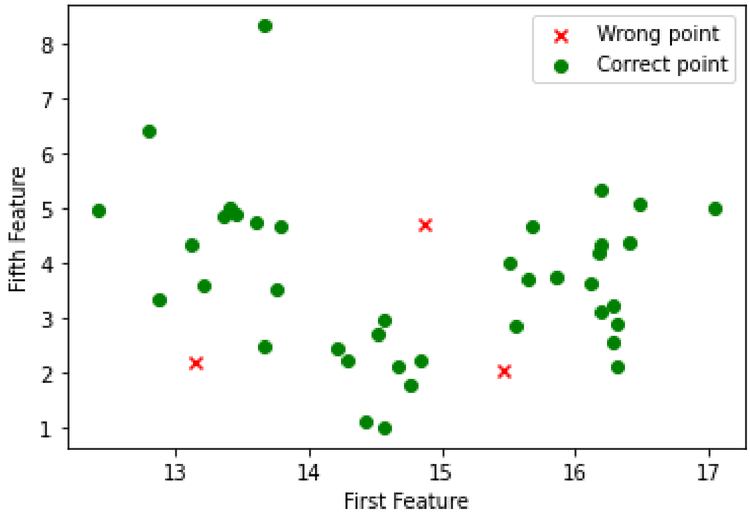




Using Argmax for Aggregation:

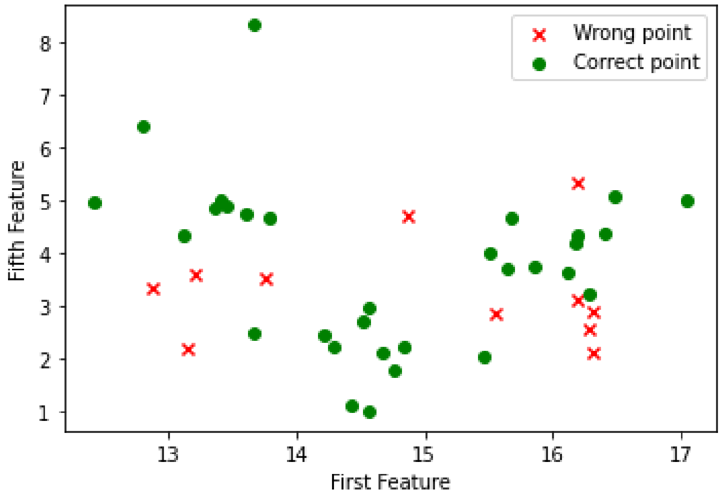
Argmax SVM Result in:

- `Accuracy: 92.857142857142 86`
- Correct and wrong prediction points plot as Shown :



Argmax Perceptron Result in:

- `Accuracy: 73.80952380952381`
- Correct and wrong prediction points (Matched= 31,Not Matched= 11) as Shown :



As shown above performance of SVM is better than Perceptron in using Argmax as An aggregation method.

Our custom Aggregation function:

Our strategy is to focus on the value returned from each classifier and use it in aggregation so

Our steps :

1- **First prepare the output of each classifier and collect them together as shown :**

```
[[1, -1, -1], [-1, 1, -1], [-1, 1, -1], [1, -1, -1], [-1,
-1, -1], [-1, 1, -1], [1, -1, 1], [-1, 1, -1], [-1, -1,
1], [-1, -1, 1]]
```

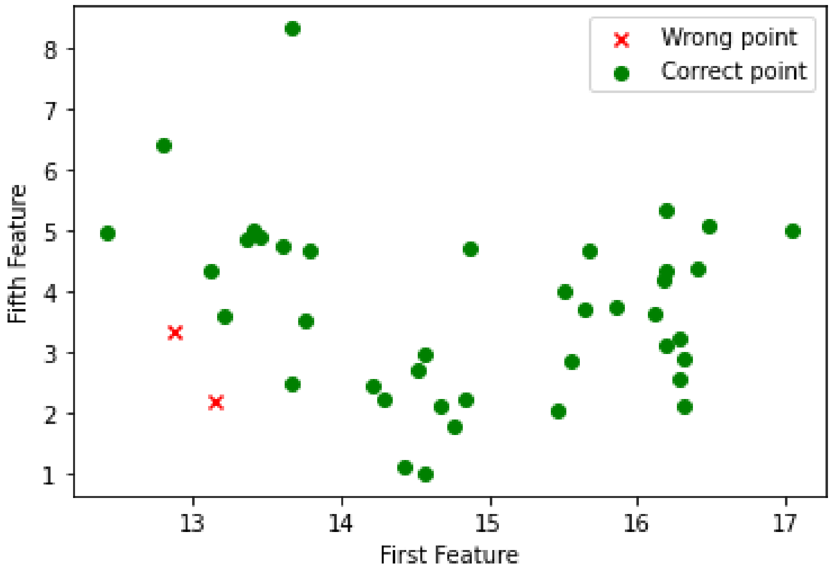
2- Second there is more scenario to select correct classifier for each feature:

- If [1,-1,-1] this means classifier one say that this belongs to him.
  Dealing with this by selecting class one
- [-1,-1,-1] all classifiers doesn't know this feature. Dealing with this by random select one of the three classes and this gives the probability of 0.33333 to select correct class
- [1,1,-1] or [1,1,1] if there are more than one classifiers say that this belongs to me. By random select between them and if there is two it gives a probability of 0.5 to be correct and with 3, probability will be 0.33333

Applying our custom Aggregation function:

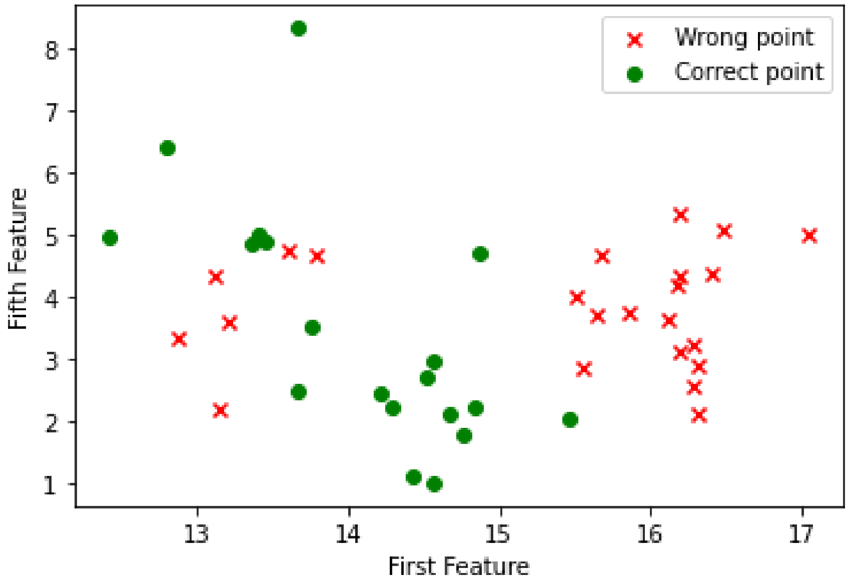
1- SVM

```
Matched= 40

Not Matched= 2

SVM_accuracy: 95.238
```



2- Perceptron

```
Matched= 19
Not Matched= 23
perception accuracy: 45.238
```



Code of Our Aggregation Function:

```
def myAggregStrat(data):
  result = []
  listOfIndexSame = []
  # loop to all values and get max value of data
  for i,value in enumerate(data):
    max_value = max(value)
    max_index = value.index(max_value)
    y = value.index(max_value, max_index,len(value))
    if max_index != y:
      listOfIndexSame.append(y)
      listOfIndexSame.append(max_index)
      # choose random item of in case of [1,1,-
1] or any number of redundant max values
      result.append( random.choice(listOfIndexSame))
    else:
      # append index of selected  to result
      result.append(max_index)
  print(result)
  for i,value in enumerate(result):
    # filter values in case of [-1,-1,-
1] and select random one from array
    if value == -1:
      result[i] = random.randint(0,2)
  return result
```

Conclusion:

- Models (Perceptron and SVM)
  Some model can give good prediction in data and other model give less prediction and in our data the SVM give us good prediction (100 %) than perceptron (83 %)
- OvR strategy :
  We learn that how to make binary classifier on data to build a model to make classification to a class versus all classes and get different prediction .
- Argmax :
  Help us to improve the high performance and improve the accuracy
- Our Aggregation Strategy :
  Our Strategy performs well with SVM , It needs more updates to be applied to Perceptron Model to achieve better accuracy.